**2019년 2학기**

**프로그래밍과 문제해결**

**Assignment #2**

|  |  |
| --- | --- |
| 담당교수 | **윤은영** |
| 학번 | **20190084** |
| 학과 | **무은재학부** |
| 이름 | **권민재** |
| POVIS ID | **mzg00** |

|  |
| --- |
| **명예서약(Honor code)**  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem 1: 흑백 공 게임**

**1. 문제의 개요**

본 프로그램의 목적은 아래와 같다.

* 두 명의 플레이어 중 검은 공의 수가 더 많은 플레이어가 이기도록 한다.
* 두 플레이어는 각각 1~5개 중 무작위 개수의 검은 공을 받는다.
* 한 플레이어가 종료를 선언하면 다른 플레이어는 최대 3번의 추가 턴을 진행할 수 있다.
* 검은 공은 최대 15개 까지 가질 수 있으며, 초과 시 초과한 만큼만 가지게 된다.

이 때 사용되는 구조 차트는 아래와 같이 표현될 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

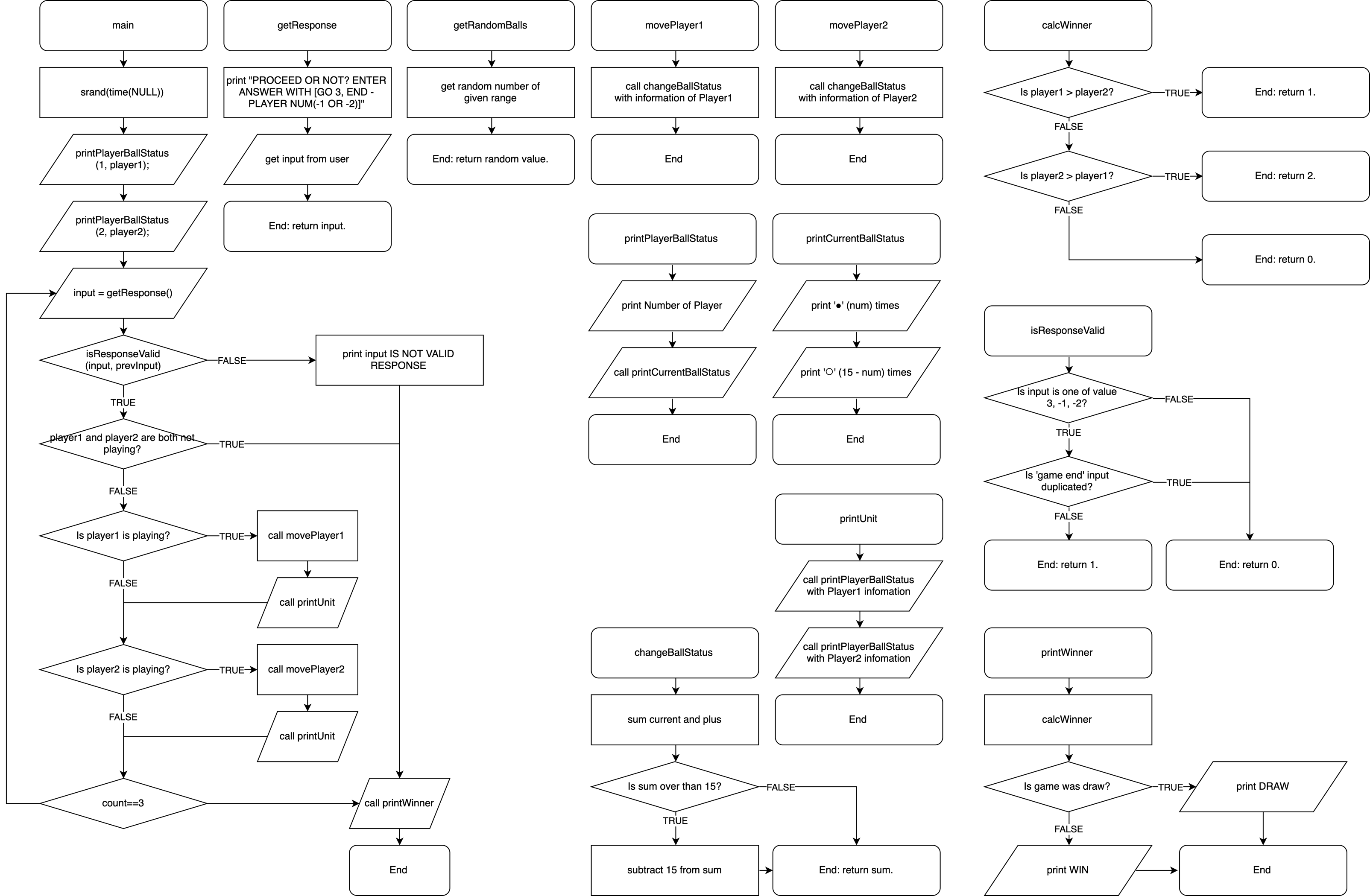
* **입력부:** 사용자의 지시를 입력받고 저장한다.
* **처리부:** 사용자의 지시가 올바른 지시인지 판단하고, 올바른 지시라면 사용자의 지시에 따라서 게임을 진행한다. 플레이어에게 무작위의 검은 공을 주거나, 플레이어가 게임을 종료했는지 등의 처리를 담당한다.
* **출력부:** 게임의 진행에 따른 공 개수와 같은 정보를 출력한다. 특히 잘못된 입력이라면 잘못된 입력이라고 출력하는 역할도 수행하여야 한다.

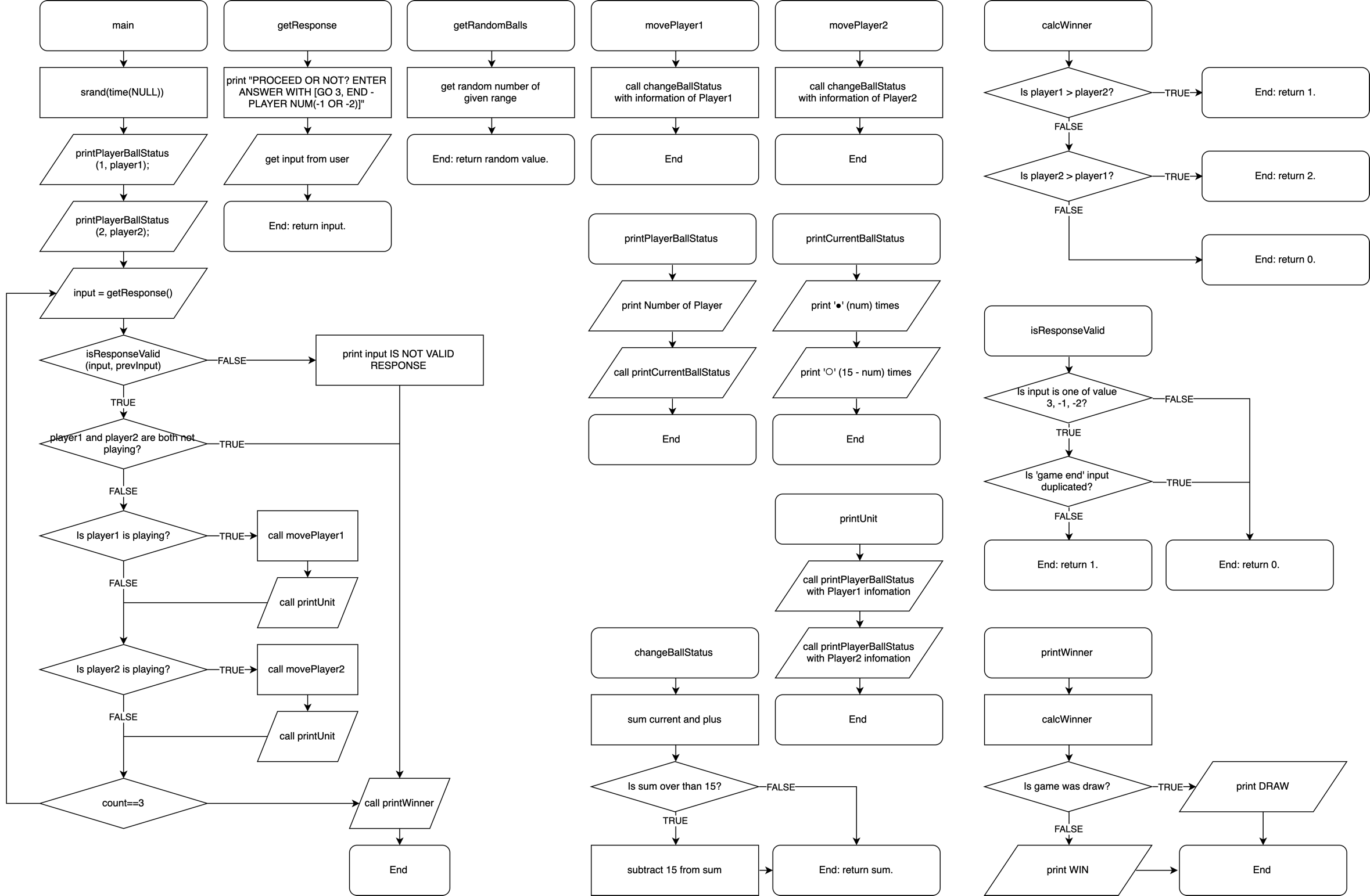
**2. 알고리즘**

해당 프로그램의 의사 코드는 아래와 같다.

|  |  |
| --- | --- |
| Pseudo-code for ASSN2\_1 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | // 프로그램에 필요한 변수들은 미리 선언한 것으로 가정하자.  include essential headers <stdio.h>, <stdlib.h>, <math.h>, <time.h>  function *getResponse*  Print “PROCEED OR NOT? ENTER ANSWER WITH [GO 3, END -PLAYER NUM(-1 OR -2)] “  Return input value from user  function *getRandomBalls*  Return random value of a given range by ‘start’ and ‘end’  function *changeBallStatus*  Sum current value and plus value.  If sum is over than 15, subtract 15 from it and return.  Else, return sum.  function *printCurrentBallStatus*  Print ‘●’ num times.  Print ‘○’ (15-num) times.  function *movePlayer1*  call *changeBallStatus* with player1 and random value obtained from *getRandomBalls*.  function *movePlayer2*  call *changeBallStatus* with player2 and random value obtained from *getRandomBalls*.  function *isResponseValid*  Check input is one of ‘3’, ‘-1’, ‘-2’  Check if game termination call is duplicated  Return 1 if input is valid, if not, return 0  function *calcWinner*  Check whose number of ball is greater.  If both number of balls are same, return 0  Else, return the number of winner player.  function *printPlayerBallStatus*  print Player number and call *printCurrentBallStatus*  function *printUnit*  print “AFTER Pn”  call *printPlayerBallStatus* with player1 and *printPlayerBallStatus* with player2  call *srand(time(NULL))* to set seed value.  call *printPlayerBallStatus* with information of player1 and player2  while  if input is valid – call *isResponseValid*  check is player1 is playing by input value.  check is player2 is playing by input value.  if player1 and player2 are both not playing, break while.    if player1 is playing, *movePlayer1* and printUnit  if player2 is playing, call *movePlayer2* and printUnit    if one player ended game, plus 1 to count. – count means `additional turn`.  if count is 3, break while.  else  print “input IS NOT VALID RESPONSE” and break while  call *printWinner.* |

위의 의사 코드를 flowchart를 이용해서 나타내면 아래와 같다.





**3. 프로그램 구조 및 설명**

**a) 사용자로부터 지시를 입력 받고 저장**

* 함수 getResponse()를 통해 사용자로부터 지시를 입력받아서 input에 저장한다.

**b) 입력 검증**

* input에 저장된 값을 isResponseValid를 통해 검증한다. isResponseValid의 반환값이 1이라면 타당한 입력이므로 게임을 진행하고, 0이면 게임 진행을 종료한다. isResponseValid는 현재 입력값과 이전 입력 값을 매개변수로 받는다. 이후, isInRange라는 변수에 연산자를 이용하여 현재 입력 값이 3, -1, -2 중 하나인지 아닌지에 따라 1 또는 0을 저장하고, inDupEnd라는 변수에 사용자의 종료 선언이 중복되었는지에 따라 1 또는 0을 저장한다. 이를 이용하여 (isInRange – isDupEnd)의 값을 반환하면 함수의 의도대로 사용자의 입력이 타당할 때 1, 아닐 때 0을 반환할 수 있다.
* 사용자의 입력이 범위 밖의 입력일 때, isInRange가 0이 되고 isDupEnd는 0일 수 밖에 없으므로 0을 반환하며, 범위 안의 입력은 맞지만 중복된 종료 선언 일 경우 isInRange가 1인데 isDupEnd가 1이므로 1 – 1 = 0을 반환할 수 있기 때문이다. 또한, 사용자의 입력이 범위 안의 입력인데 중복이 아니면, isInRange는 1이고 isDupEnd는 0이 되므로 1을 반환할 수 있음을 볼 수 있다.

**c) 플레이어의 게임 참여 여부에 따른 게임 진행**

* 각 플레이어의 게임 종료 여부를 알려주는 변수, isPlaying1과 isPlaying2을 이용하여 게임의 진행을 제어한다. isPlayingN이 0이면 플레이어 N은 게임을 종료 했음을 의미하며, 1이면 플레이어 N이 게임에 참여 중임을 의미한다. 기본적으로 두 변수는 1로 초기화 되어있다.
* 우선 입력 값이 검증된 후, 그 값이 -1이라면 isPlaying1에 0을, -2라면 isPlaying2에 0을 저장한다.
* isPlaying1과 isPlaying2가 모두 0이라면, 게임 진행을 종료한다.
* isPlaying1이 1이라면, 함수 movePlayer1을 이용하여 player1의 공의 개수를 변경하고, 함수 printUnit을 호출하여 변화된 공 개수를 출력한다.
* isPlaying2가 1이라면, 함수 movePlayer2을 이용하여 player2의 공의 개수를 변경하고, 함수 printUnit을 호출하여 변화된 공 개수를 출력한다.
* isPlaying1과 isPlaying2의 값이 다르다면, 한 플레이어가 게임을 종료한 것으로 추론할 수 있으므로, 추가 턴 진행 정도를 저장하는 변수 count에 (isPlaying1 != isPlaying2)의 값을 더한다.
* count가 3이 되면 게임 진행을 종료한다.
* 게임 진행이 종료될 때 까지 입력부터의 과정을 반복한다.

**d) 결과 출력**

* 게임 진행이 종료되면, 함수 printWinner를 이용하여 게임의 결과를 출력한다.

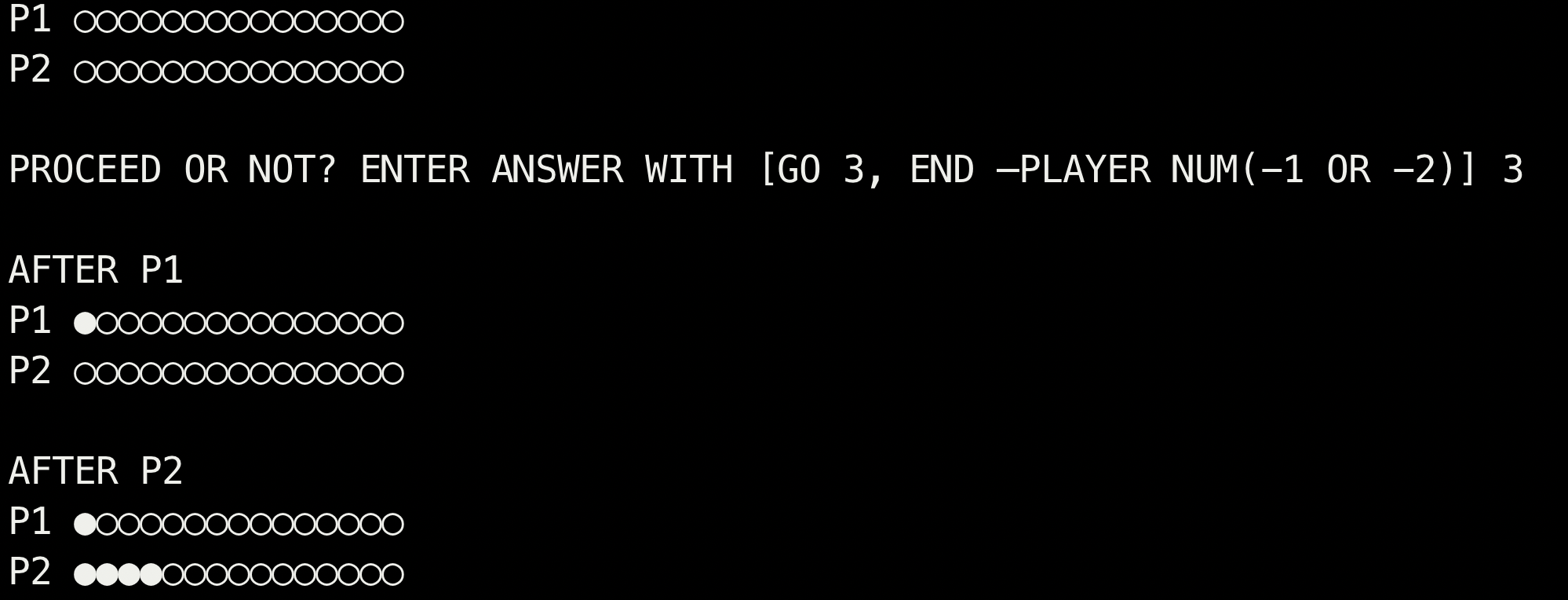
**4. 프로그램 실행 방법 및 예제**

**a) 초기 화면**

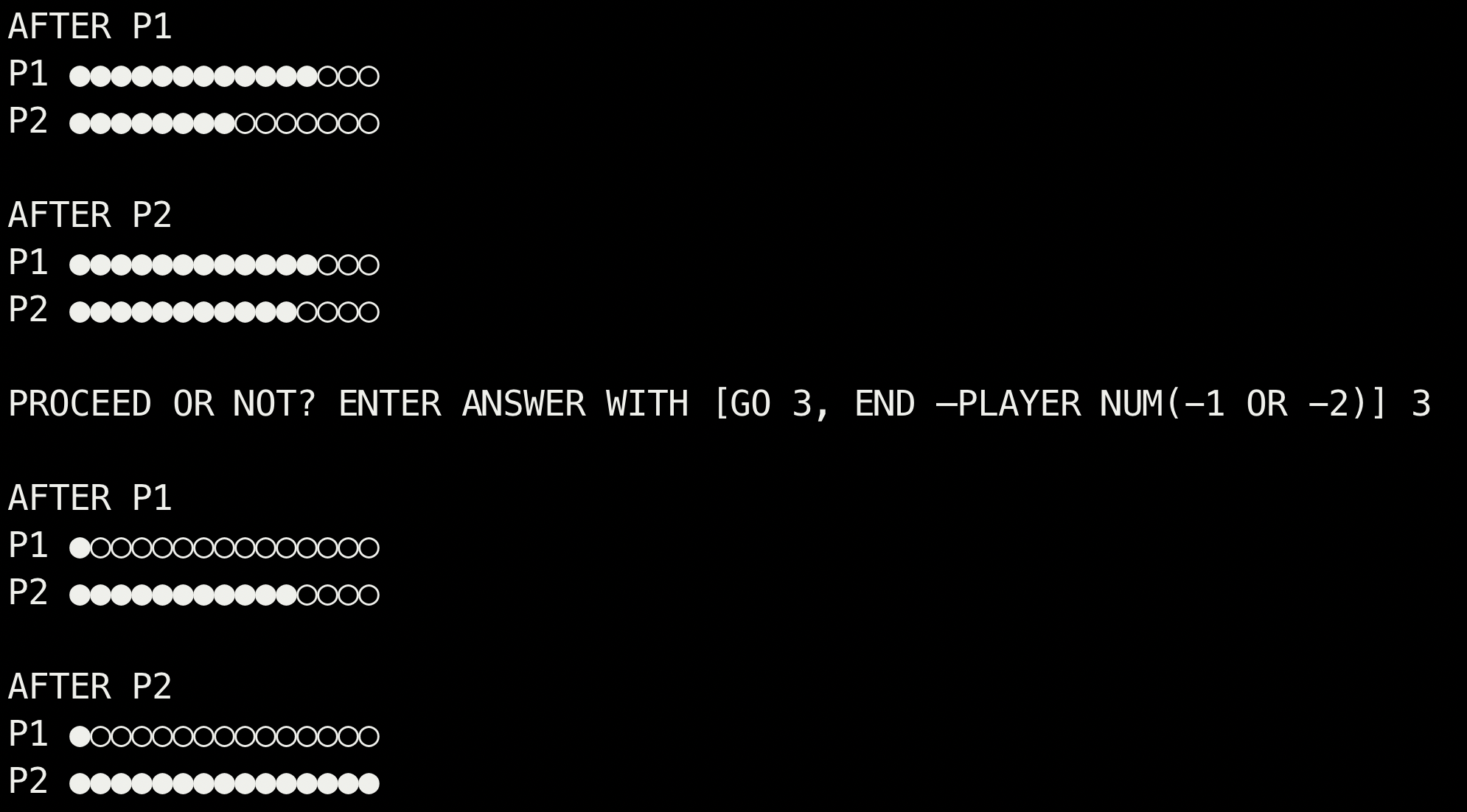


프로그램을 처음 실행하면, 각 플레이어에게 주어진 15개의 흰색 공이 출력되고, 사용자의 입력을 대기한다.

**b) 게임 계속 진행**

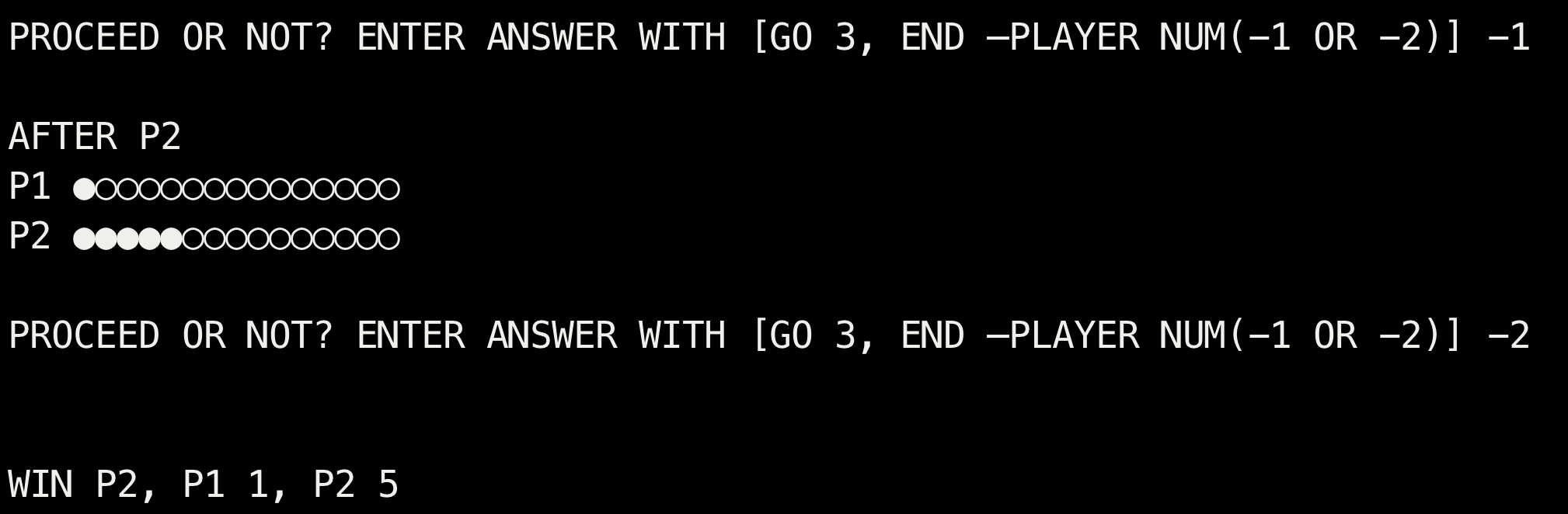


처음 3을 입력하면 각 플레이어들이 게임을 진행하는 모습을 볼 수 있다.

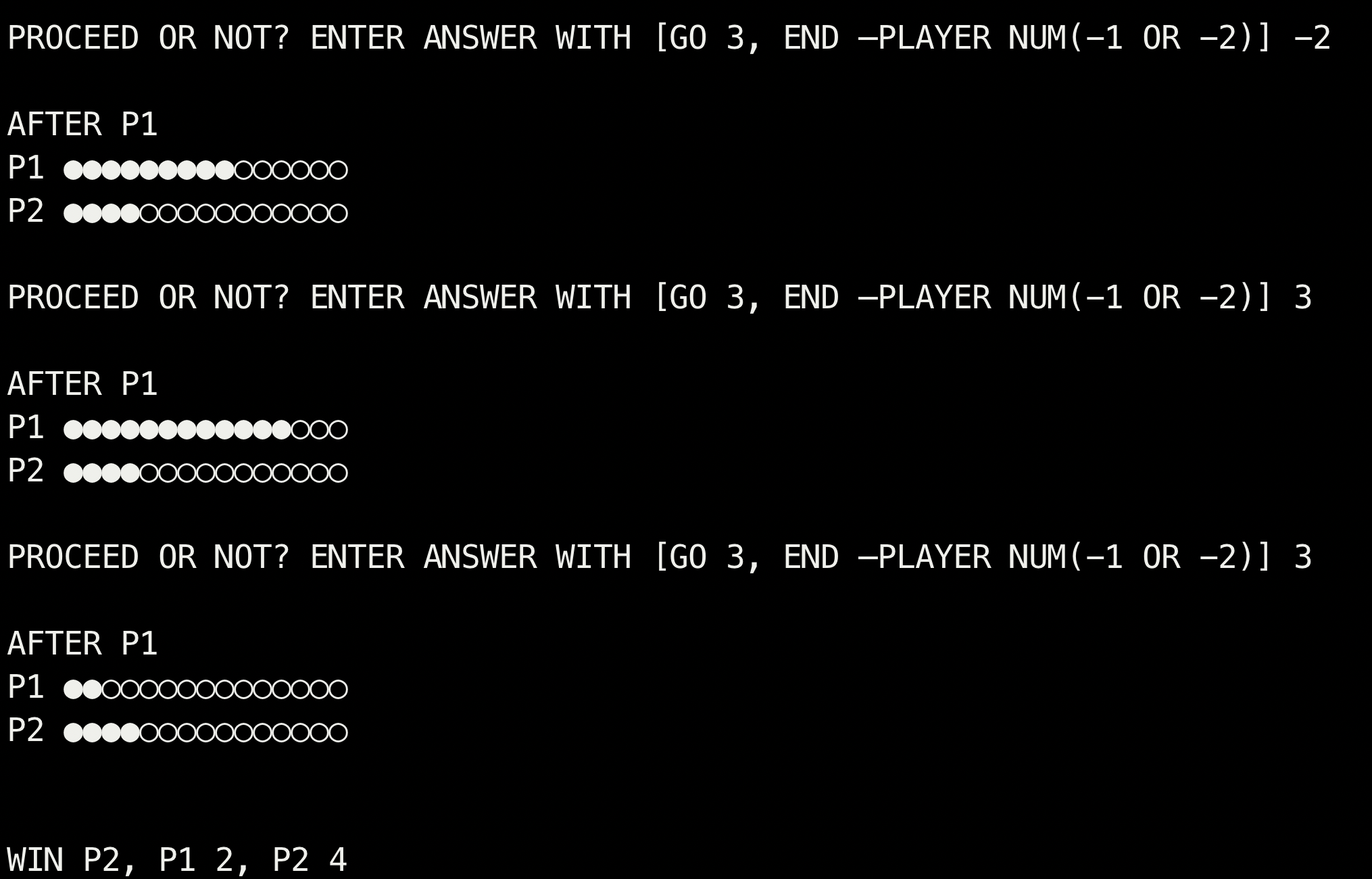


검은색 공의 개수가 15개를 초과하면 초과한 수 만큼의 검은 색 공만 가지게 됨을 플레이어 1의 결과를 통해 볼 수 있다.

**c) 게임 종료**

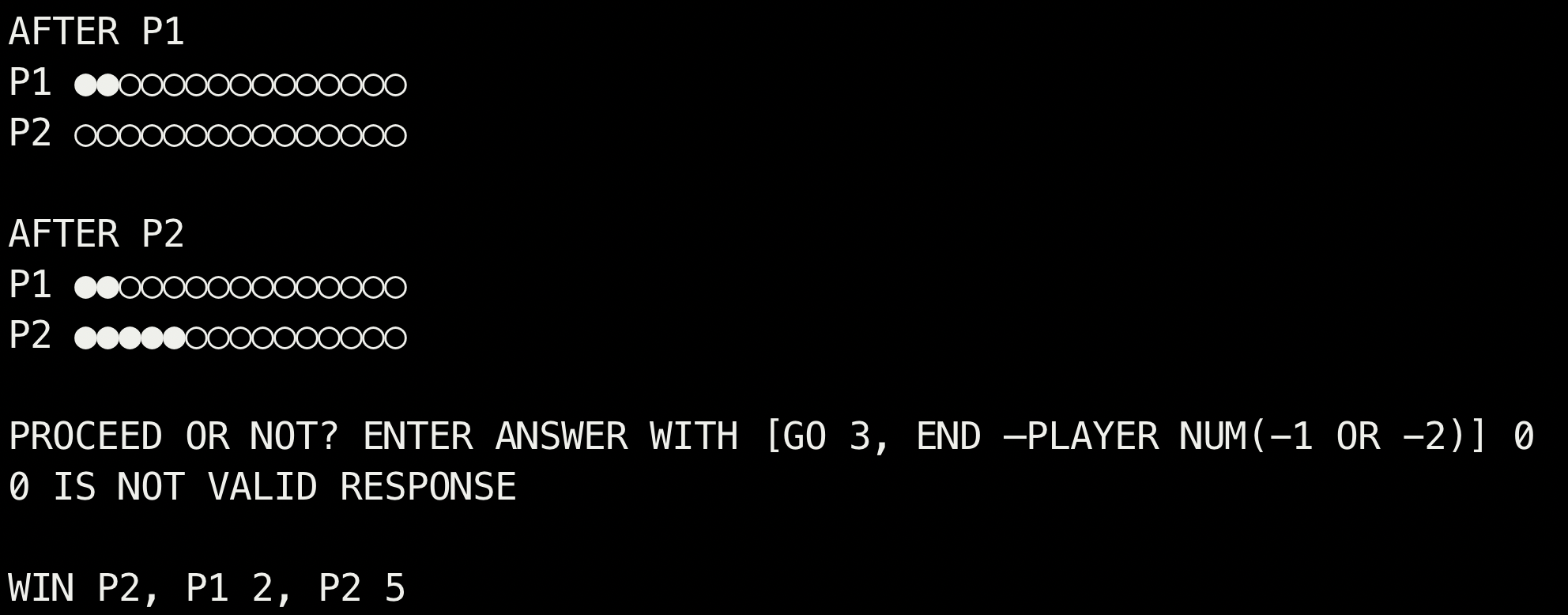


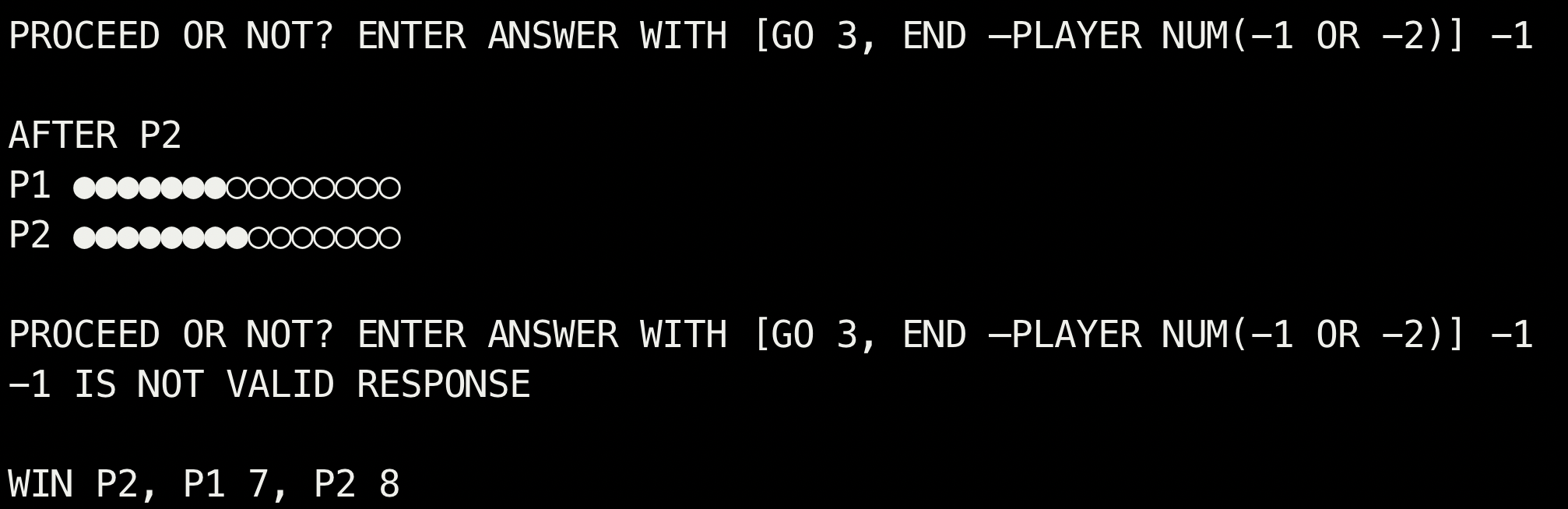
-1을 입력하고 -2를 입력하면 승패 여부가 출력되고 종료되는 모습을 볼 수 있다.



-2를 입력하고 난 후에 남은 플레이어는 최대 3턴까지 진행하고 승패 여부를 출력하고 종료되는 모습을 볼 수 있다.

**d) 유효하지 않은 입력**

****

****

3, -1, -2가 아닌 입력이나 중복된 종료 선언 일 경우 유효하지 않은 입력임을 출력하고 승패를 출력하고 종료함을 볼 수 있다.

**5. 토론**

* 주어진 문제에서는 rand()를 이용해서 난수를 발생시키고 있는데, 사실 이는 예측 가능한 난수이다. rand()를 이용해서 구하는 난수를 그래서 특히 pseudo random, 즉 유사 난수라고 부르며, 같은 시드 값만 있다면 같은 난수를 얻을 수도 있다. 유사 난수를 통해 꽤나 고르게 분포된 난수를 얻을 수는 있겠지만, 완벽한 난수를 얻을 수는 없다. 이러한 유사 난수를 발생시키는 방법으로는 중앙제곱법, 메르센 트위스터 등 여러 방법이 존재한다.

**6. 결론**

* 본 과제를 통해 랜덤 함수는 임의로 시드를 설정한 다음 써야 한다는 것을 익힐 수 있었고, 조건문과 반복문을 이용하여 주어진 문제를 프로그래밍을 이용하여 쉽게 풀 수 있다는 사실을 알 수 있었다.

**7. 개선방향**

* 현재 작성된 코드에서는 movePlayer1과 movePlayer2가 문제에서 주어진대로 player1과 player2, 2개의 변수를 입력받는데, 해당 함수들 안에서는 1개의 변수만 사용하고 있다. 1개의 변수만 사용하도록 개선할 필요가 있다.

**Problem 2: 볼링 점수 계산**

**1. 문제의 개요**

본 프로그램의 역할은 아래와 같다.

* 파일로부터 볼링의 점수 기록을 입력받는다.
* 입력 받은 기록을 바탕으로 볼링 점수를 계산한다.
* 계산한 결과를 출력한다.

이 때 사용되는 구조 차트는 아래와 같이 표현될 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

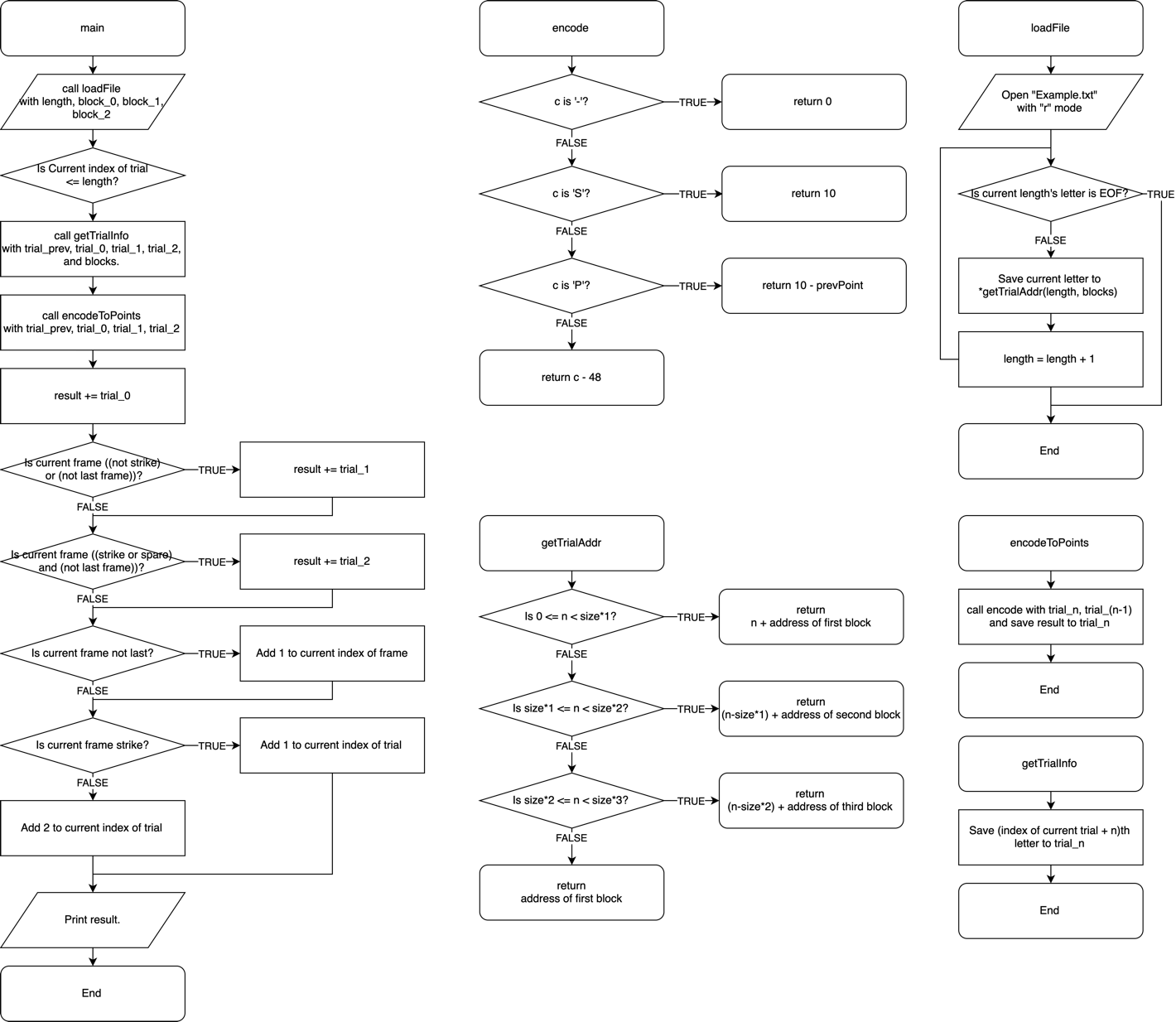
* **입력부:** 파일로부터 볼링 점수 기록을 입력 받아서 저장한다.
* **처리부:** 저장한 점수 기록을 바탕으로 볼링의 프레임 별 기본 점수와 추가 점수를 고려하여 점수의 총합을 계산한다.
* **출력부:** 구한 점수 총합을 출력한다.

**2. 알고리즘**

해당 프로그램의 의사코드는 아래와 같다.

|  |  |
| --- | --- |
| Pseudo-code for measure\_distance | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | // 프로그램에 필요한 변수들은 미리 선언한 것으로 가정하자.  include essential headers <stdio.h  function *getTrialAddr*  Save addresses of three blocks (block\_0, block\_1, block\_2) from main function to char pointer. // Use char\* (char-pointer) to read/write block in 1 bytes.  Get n, which is index we need.  Get size, which is byte-size of block.  If 0 <= n < size\*1, return (n + address of first block).  If size\*1 <= n < size\*2, return ((n-size\*1) + address of second block).  If size\*2 <= n < size\*3, return ((n-size\*2) + address of third block).  Else, return address of first block. – it’s for when n is -1.  function *loadFile*  Get addresses of three blocks (block\_0, block\_1, block\_2) from main function.  Open “Example.txt” with “r” mode.  While letter of “Example.txt” is not EOF, save letters to block\_1, block\_2, block\_3 using *getTrialAddr*  Save length of text file to variable of main function.  Close file stream of “Example.txt”.  function *getTrialInfo*  Get index of current trial, idx\_trial.  Get addresses of trial\_prev, trial\_0, trial\_1, trial\_2. // trial\_n means information of (idx\_trial + n)th trial. ‘prev’ is -1.  Save (index of current trial + n)th letter to trial\_n using *getTrialAddr*.  function *encode*  If c is ‘-‘, return 0. Else if c is ‘S’, return 10. Else if c is ‘P’, return (10 – prevPoint).  Else, return (c – 48). // c is letter, so we need to subtract 48 to get number.  function *encodeToPoints*  Get addresses of trial\_prev, trial\_0, trial\_1, trial\_2.  Encode trial\_n to point using function *encode* and save to it.  Call *loadFile* to save text file to block\_0, block\_1, block\_2 and get length of text file.  While current index of trial is less than length of text file,  // trial\_n is (current index + n)th letter information of text file.  Call *getTrialInfo* to save trial\_prev, trial\_0, trial\_1, trial\_2 from blocks  Call *encodeToPoints* to encode trial\_n to points.    Add trial\_0 to result.  If current frame is not strike or last frame, add trial\_1 to result.  If current frame is (strike or spare), and not last frame, add trial\_2 to result.    If current frame is not last, add 1 to current index of frame.  If current frame is strike, add 1 to current index of trial, else add 2 to it.  Print result. |

위의 의사 코드를 Flowchart를 이용해서 나타내면 아래와 같다.



**3. 프로그램 구조 및 설명**

**a) 데이터 관리 체계**

* 이 프로그램에서 가장 중요한 부분은 ‘파일에서 읽은 데이터를 어떻게 관리하느냐’이다. 데이터의 효율적인 관리를 위해, 해당 부분을 함수 *getTrialAddr*로 관리한다. *getTrialAddr*은 block\_0, block\_1, block\_2의 주소와 접근하고자 하는 글자의 인덱스 n을 매개변수로 가지고, char형 포인터를 반환하는, 즉 주소를 반환하는 함수이다.
* *getTrialAddr*의 작동방식을 이해하기 위해서는 포인터에 대한 생각이 필요하다. 포인터를 이용해 값을 역참조하는 과정을 잘 관찰해보았을 때, int형 포인터를 역참조하면 컴파일러는 자동으로 int의 크기만큼 역참조 하고, char형 포인터를 역참조하면 컴파일러는 자동으로 char의 크기만큼 역참조 한다고 생각할 수 있다. 그렇다면, long long int형 포인터를 char형 포인터로 역참조하면 char 자료형의 크기 만큼만 가져올 것이라고 생각할 수 있다.
* 위의 특성을 이용하면, char형 포인터를 이용하여 long long int 형식인 block들을 1바이트씩 읽고 쓸 수 있다.
* long long int의 바이트 크기를 size라고 할 때, 이제 1바이트 씩 읽고 쓰는 작업이 가능하므로, 각 block에서 size개의 글자들을 처리할 것이며, block\_0부터 1개씩 채워나가는 식으로 순서를 매길 것이다.
* 이때, 첫 인덱스를 0이라고 생각하고, 블럭 각각의 가장 낮은 주소를 담고 있는 char형 포인터를 addr\_block\_0, addr\_block\_1, addr\_block\_2라고 하자. 그럼 n < size일 때 addr\_block\_1 + n, size <= n < 2\*size일 때 addr\_block\_2 + (n-size), 2\*size <= n < 3\*size일 때 addr\_block\_3 + (n – 2\*size)를 반환함으로서 block들 안에서 n번째 글자가 담겨있는 곳의 주소를 반환할 수 있고, 이를 이용해 block들에 저장된 글자들 중에서 원하는 인덱스의 글자를 읽고 쓸 수 있다.

**b) 파일 입력 및 저장**

* 함수 *loadFile*에서 fopen을 통해 “Example.txt”를 “r”, 즉 읽기 모드로 연 후, *getTrialAddr*을 이용해서 글자들을 block\_0, block\_1, block\_2에 저장한다. EOF가 나기 전까지 반복하는데 ,저장할 때 마다 저장하는 인덱스는 1씩 증가시키며, 반복이 종료된 후 인덱스는 텍스트 파일의 글자 길이라고 생각할 수 있다. 이 길이 값은 main에서 사용한다.

**c) 최종 점수 계산**

* 최종 점수 계산은 인덱스를 기준으로 생각한다. idx\_frame은 현재 고려 중인 frame의 인덱스이며, idx\_trial은 현재 고려중인 trial[[1]](#footnote-1)의 인덱스를 말한다.
* 함수 *getTrialInfo*를 이용해서 trial\_prev, trial\_0, trial\_1, trial\_2에 각각 idx\_trial - 1, idx\_trial, idx\_trial + 1, idx\_trial + 2번째 글자를 저장한다.
* 현재 고려 중인 프레임이 스트라이크인지 아닌지의 여부를 isStrike, 스페어인지 아닌지의 여부를 isSpare, 마지막 프레임이 아닌지 맞는지의 여부를 isNotLast에 각각 1이나 0으로 저장한다. trial\_0이 ‘S’라면, isStrike가 1이 될 것이고, trial\_1이 ‘P’라면 isSpare가 1이 될 것이다. 그리고 idx\_frame은 0부터 시작하므로, idx\_frame이 9가 아니라면 isNotLast가 1이 될 것이다.
* 함수 *encodeToPoints*를 이용하여 trial\_prev, trial\_0, trial\_1, trial\_2에 담긴 값을 점수로 변환한다.
* 문제에서 주어진 함수 *encode*는 *encodeToPoints*에서 사용되는데, 다른 변수들과의 자료형을 맞춰주기 위해, char c를 int c로 수정하였다.
* 함수 encode는 c가 ‘-‘이면 0, ‘S’이면 10, ‘P’이면 10-prevPoint을 반환하는데, 모두 아닌 경우에는 c – 48을 반환한다. 이때, 48을 빼는 이유는, c가 아스키코드 형식이기 때문에 실제 의미하는 숫자로 바꿔주기 위함이다.
* 최종 결과 점수에 trial\_0은 무조건 더해져야 한다. trial\_1은 스트라이크가 아니거나, 마지막 라운드가 아닌 상황이면 최종 결과 점수에 더해져야 한다. 현재 프레임이 스트라이크라면, trial\_1을 첫번째 추가 점수로 간주할 수 있기 때문이다. trial\_2는 현재 프레임이 스트라이크나 스페어일 때, 마지막 프레임이 아니면 최종 결과 점수에 더해져야 한다. 이때, trial\_2는 스페어인 상황에서는 첫번째 추가 점수, 스트라이크인 상황에서는 두번째 추가 점수로 간주할 수 있다. 이를 간결하게 표현하면, 최종 점수에 trial\_0 + ((!isStrike) || isNotLast) \* trial\_1 + ((isStrike || isSpare) && isNotLast) \* trial\_2이 더해져야 하는 것으로 표현할 수 있다.
* 이 프레임이 마지막 프레임이 아니라면 idx\_frame에 1을 더하고, idx\_trial에는 2 – isStrike를 더한 뒤, idx\_trial이 글자 길이 이하 일 때 까지 계속 반복해서 최종 점수 구하는 과정을 반복한다.

**d) 최종 점수 출력**

* printf로 최종 결과를 출력한다.

**4. 프로그램 실행 방법 및 예제**

프로그램을 실행하면 아래와 같이 입력된 내용과 각 예시에 맞는 올바른 점수가 출력되는 것을 볼 수 있다.

|  |  |
| --- | --- |
|  | Example1 출력 결과 |
|  | Example2 출력 결과 |
|  | Example3 출력 결과 |
|  | Example4 출력 결과 |

**5. 토론**

* getTrialAddr을 잘 설계한다면, 이 함수가 필요로 하는 변수들의 주소 목록도 getTrialAddr을 이용하여 불러올 수 있을 것이다. 즉, getTrialAddr을 재귀함수로 쓰는 것이다.
* getTrialAddr에서 size를 8로 고정하지 않은 이유는, 혹시 모를 컴파일러 간의 격차를 줄이기 위함이다. 특정 컴파일러에서는 long long int 자료형의 크기가 8 바이트가 아닐 수도 있기 때문이다. 하지만 만약 long long int 자료형의 크기가 8바이트보다 작다면, 볼링 점수 데이터를 모두 담을 수 없기 때문에, 문제가 발생할 것이다.
* 모두 문자 데이터이기에 char형 자료형으로 모두 처리할 수 있지만, 혹시 모를 경우를 대비해서 그보다 큰 자료형인 int를 주로 사용하였다.

**6. 결론**

* 본 과제를 통해 포인터의 작동 원리에 대해 깊게 고찰해 볼 수 있었고, 다양한 사용자 정의 함수를 구현하여 문제를 해결하는 능력을 키울 수 있었다.

**7. 개선방향**

* 전처리기를 통한 매크로를 “#define L(x) \*(getTrialAddr(x, &block\_1, &block\_2, &block\_3))”과 같이 표현하여 이용하면 더 간결하게 사용할 수 있다.
* 차후에 배울 배열을 이용하면 더 깔끔하게 이 문제를 해결할 수 있을 것이라고 생각한다.

1. 한 프레임 내에서 한 번 던진 시기. [↑](#footnote-ref-1)